CHAPTER 7

Data Visualization with Python Libraries

In the last chapter, we read about Pandas, the library with various functions for preparing data in order to make it ready for analysis and visualization. Visualization is a means to understand patterns in your data, identify outliers and other points of interest, and present our findings to an outside audience, without having to sift through data manually. Visualization also helps us to glean information from raw data and gain insights that would otherwise be difficult to draw.

After going through this chapter, you will be able to understand the commonly used plots, comprehend the object-oriented and stateful approaches in Matplotlib and apply these approaches for visualization, learn how to use Pandas for plotting, and understand how to create graphs with Seaborn.

Technical requirements

In your Jupyter notebook, type the following to import the following libraries.

CODE:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Here, *plt* is a shorthand name or an alias for the *pyplot* module of Matplotlib that we use for plotting, *sns* is an alias for the Seaborn library, and *pd* is an alias for Pandas.

© Gayathri Rajagopalan 2021
G. Rajagopalan, *A Python Data Analyst's Toolkit*, https://doi.org/10.1007/978-1-4842-6399-0_7

In case these libraries are not installed, go to the Anaconda Prompt and install them as follows:

```
pip install matplotlib
pip install seaborn
pip install pandas
```

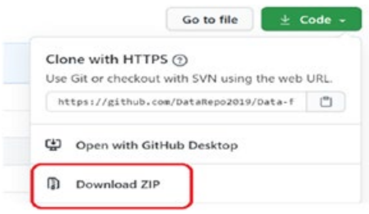
External files

We use the *Titanic* dataset in this chapter to demonstrate the various plots.

Please download the dataset using the following link: `https://github.com/DataRepo2019/Data-files/blob/master/titanic.csv`

You can also download this dataset using the following steps:

- Click the following link: `https://github.com/DataRepo2019/Data-files`
- Click: Code ➤ Download ZIP

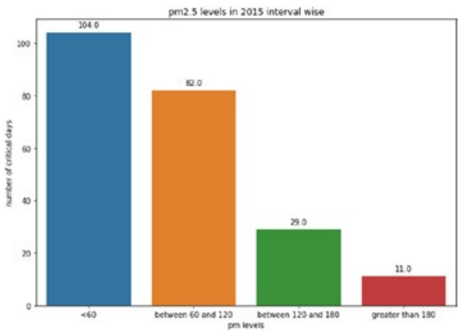
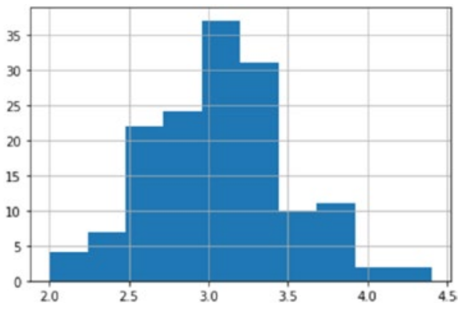

Go to file
Code
Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/DataRepo2019/Data-f
Open with GitHub Desktop
Download ZIP

- From the downloaded zip folder, open the “titanic.csv” file

Commonly used plots

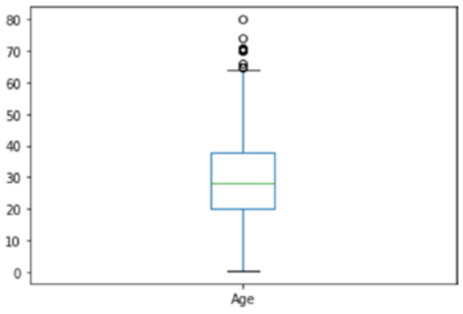
Some of the basic plots that are widely used in exploratory or descriptive data analysis include bar plots, pie charts, histograms, scatter plots, box plots, and heat maps; these are explained in Table 7-1.

Table 7-1. *Commonly Used Plots to Visualize Data in Descriptive Data Analysis*

Type of Chart or Plot	Description	Shape
Bar chart	A bar chart enables visualization of categorical data, with the width or height of the bar representing the value for each category. The bars can be shown either vertically or horizontally.	
Histogram	A histogram is used to visualize the distribution of a continuous variable. It divides the range of the continuous variable into intervals and shows where most of the values lie.	

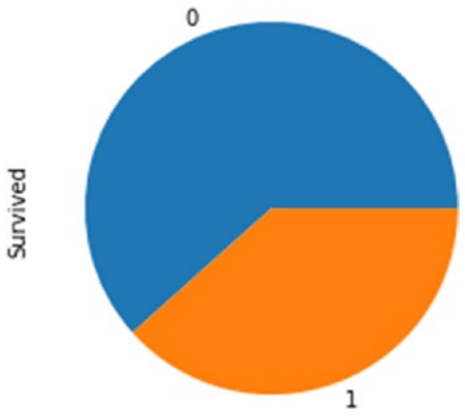
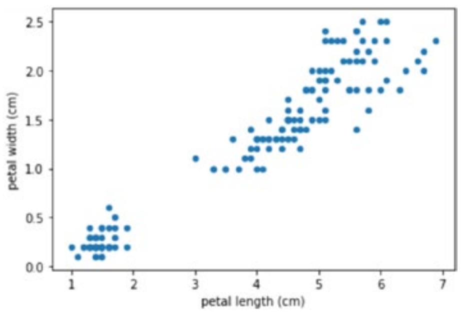
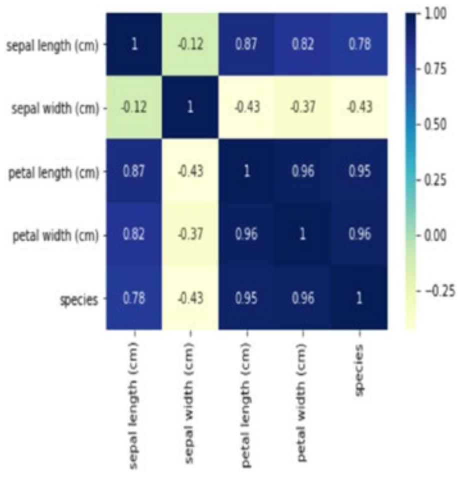
(*continued*)

Table 7-1. *(continued)*

Type of Chart or Plot	Description	Shape
Box plots	Box plots help with visually depicting the statistical characteristics of the data. A box plot provides a five-point summary with each line in the figure representing a statistical measure of the data being plotted (refer to the figure on the right). These five measures are • Minimum value • 25th percentile • Median (50th percentile) • 75th percentile • Maximum value The small circles/dots that you see in the figure on the right represent the outliers (or extreme values). The two lines on either side of the box, representing the minimum and maximum values, are also called "whiskers". Any point outside these whiskers is called an outlier. The middle line in the box represents the median. A box plot is generally used for continuous (ratio/interval) variables, though it can be used for some categorical variables like ordinal variables as well.	

(continued)

Table 7-1. *(continued)*

Type of Chart or Plot	Description	Shape
Pie charts	A pie chart shows the distinct values of a variable as sectors within a circle. Pie charts are used with categorical variables.	
Scatter plots	A scatter plot displays the values of two continuous variables as points on the x and y axes and helps us visualize if the two variables are correlated or not.	
Heat maps	A heat map shows the correlation between multiple variables using a color-coded matrix, where the color saturation represents the strength of the correlation between the variables. A heat map can aid in the visualization of multiple variables at once.	

Let us now have a look at some of the Python libraries that are used for visualization, starting with Matplotlib.

Matplotlib

The main library for data visualization in Python is Matplotlib. Matplotlib has many visualization features that are similar to Matlab (a computational environment cum programming language with plotting tools). Matplotlib is mainly used for plotting two-dimensional graphs, with limited support for creating three-dimensional graphs.

Plots created using Matplotlib require more lines of code and customization of the parameters of the plot, as compared to other libraries like Seaborn and Pandas (which use some default setting to simplify the writing of code to create plots).

Matplotlib forms the backbone of most of the visualizations that are performed using Python.

There are two interfaces in Matplotlib, stateful and object-oriented, that are described in Table 7-2.

***Table 7-2.** Interfaces in Matplotlib*

Stateful interface	Object-oriented interface
This interface uses the *pyplot* class, which is based on Matlab. A single object of this class is created, which is used for all plotting purposes.	In this interface, we use different objects for different elements of the plot. The two main objects that are used in this interface for plotting are • The *figure* object, which acts as the container for other objects. • The *axes* object, which is the actual plotting area containing the x axis, y axis, points, lines, legends, and labels. Note that the axes here does not refer to the x and y axes but the entire subplot.
Code Example (visualization using stateful interface):	Code Example (visualization using object-oriented interface):
`import matplotlib.pyplot as plt`	`import matplotlib.pyplot as plt`
`%matplotlib inline`	`%matplotlib inline`
`x=[1,2,3,4]`	`x=[1,2,3,4]`
`y=[3,6,9,12]`	`y=[3,6,9,12]`
`plt.plot(x,y) # The plot function plots a line between the x and y coordinates`	`fig,ax=plt.subplots(figsize=(10,5)) #The subplots method creates a tuple returning a figure and one or more axes.`
`plt.xlim(0,5) # Sets limits for the x axis`	`ax.plot(x,y) #Creating a plot with an axes object`
`plt.ylim(0,15) # Sets limits for the y axis`	

(*continued*)

Table 7-2. (*continued*)

Stateful interface	Object-oriented interface

`plt.xlabel('X axis') #labels the x axis`
`plt.ylabel('Y axis') #labels the y axis`
`plt.title('Basic plot') #Gives a title`
`plt.suptitle('Figure title',size=20,y=1.02) # Gives a title to the overall figure`	

Customization (with the stateful interface): In this interface, all changes are made using the current state of the pyplot object that points to the figure or axes, as shown in the following.	Customization (with the object-oriented interface): In this interface, since we have different objects for the figure and each of the subplots or axes, these objects are customized and labeled individually, as shown in the following.

Code Example:
`#This code makes changes to the graph created using the plt object`
`ax=plt.gca() # current axes`
`ax.set_ylim(0,12) #use it to set the y axis limits`
`fig=plt.gcf() #current figure`
`fig.set_size_inches(4,4) #use it to set the figure size`
	Code Example:
	`#this code makes changes to the graph created using the preceding object-oriented interface`
	`ax.set_xlim(0,5) # Sets limit for the x axis`
	`ax.set_ylim(0,15) # Sets limit for the y axis`
	`ax.set_xlabel('X axis') #Labels x axis`
	`ax.set_ylabel('Y axis') #Labels y axis`
	`ax.set_title('Basic plot') # Gives a title to the graph plotted`
	`fig.suptitle('Figure title',size=20,y=1.03) #Gives a title to the overall figure`

Further reading: See more about these two different interfaces: https://matplotlib.org/3.1.1/tutorials/introductory/lifecycle.html

Approach for plotting using Matplotlib

The object-oriented approach is the recommended approach for plotting in Matplotlib because of the ability to control and customize each of the individual objects or plots. The following steps use the object-oriented methodology for plotting.

1. **Create a figure (the outer container) and set its dimensions**:

   The *plt.figure* function creates a figure along with setting its dimensions (width and height), as shown in the following.

   CODE:

   ```
   fig=plt.figure(figsize=(10,5))
   ```

2. **Determine the number of subplots and assign positions for each of the subplots in the figure**:

   In the following example, we are creating two subplots and placing them vertically. Hence, we divide the figure into two rows and one column with one subplot in each section.

   The *fig.add_subplot* function creates an axes object or subplot and assigns a position to each subplot. The argument –211 (for the *add_subplot* function that creates the first axes object - "ax1") means that we are giving it the first position in the figure with two rows and one column.

   The argument -212 (for the *add_subplot* function that creates the second axes object - "ax2") means that we are giving the second position in the figure with two rows and one column. Note that the first digit indicates the number of rows, the second digit indicates the number of columns, and the last digit indicates the position of the subplot or axes.

CODE:

```
ax1=fig.add_subplot(211)
ax2=fig.add_subplot(212)
```

3. **Plot and label each subplot**:

   After the positions are assigned to each subplot, we move on to generating the individual subplots. We are creating one histogram (using the *hist* function) and one bar plot (using the *bar* function). The x and y axes are labeled using the *set_xlabel* and *set_ylabel* functions.

   CODE:

```
labelling the x axis

ax1.set_xlabel("Age")

#labelling the yaxis

ax1.set_ylabel("Frequency")

#plotting a histogram using the hist function

ax1.hist(df['Age'])

#labelling the X axis

ax2.set_xlabel("Category")

#labelling the Y axis

ax2.set_ylabel("Numbers")

#setting the x and y lists for plotting the bar chart

x=['Males','Females']

y=[577,314]

#using the bar function to plot a bar graph

ax2.bar(x,y)
```

Output :

Note that the top half of Figure 7-1 is occupied by the first axes object (histogram), and the bottom half of the figure contains the second subplot (bar plot).

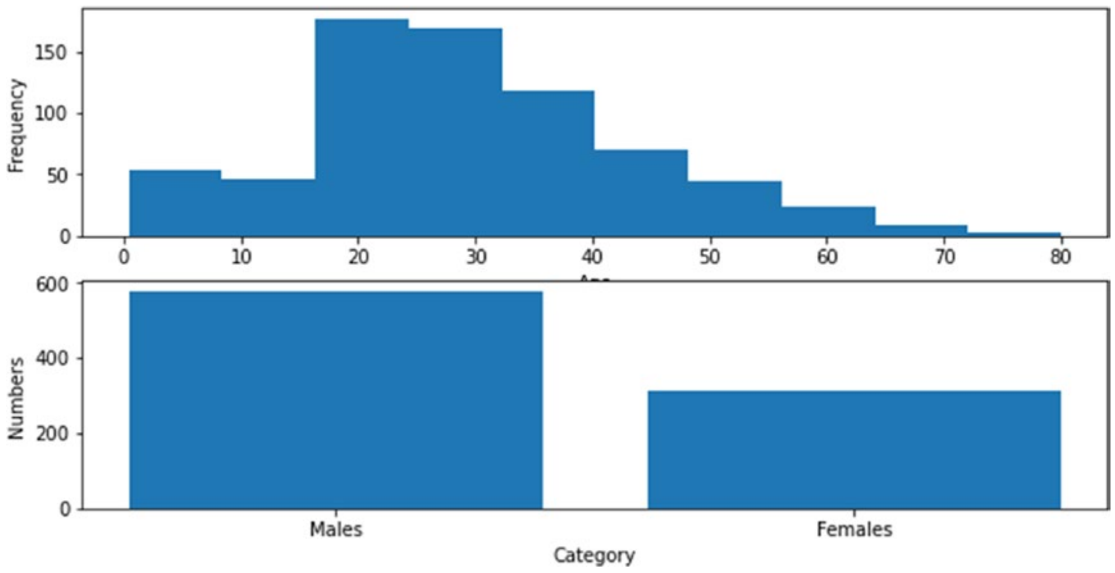


Figure 7-1. *Subplots within a figure*

Plotting using Pandas

The Pandas library uses the Matplotlib library behind the scenes for visualizations, but plotting graphs using Pandas functions is much more intuitive and user-friendly. Pandas requires data to be in the wide or aggregated format.

The *plot* function (based on the Matplotlib plot function) used in Pandas allows us to create a wide variety of plots simply by customizing the value of the *kind* parameter, which specifies the type of plot. This is also an example of polymorphism in object-oriented programming (one of the principles of OOPS, which we studied in Chapter 2), where we are using the same method for doing different things. The *kind* parameter in the *plot* method changes with the kind of graph you want to plot.

Let us learn how to create plots in Pandas using the *Iris* dataset.

The *Iris* dataset contains samples from various species of the iris plant. Each sample contains five attributes: sepal length, sepal width, petal length, petal width, and species (*Iris setosa*, *Iris versicolor*, and *Iris virginica*). There are 50 samples of each species. The *Iris* dataset is inbuilt in the *sklearn* library and can be imported as follows:

CODE:

```
import pandas as pd
from sklearn.datasets import load_iris
data=load_iris()
iris=pd.DataFrame(data=data.data,columns=data.feature_names)
iris['species']=pd.Series(data['target']).map({0:'setosa',1:'versicolor',2:
'virginica'})
```

Scatter plot

A scatter plot helps us understand if there is a linear relationship between two variables. To generate a scatter plot in Pandas, we need to use the value *scatter* with the parameter *kind* and mention the columns (specified by the parameters "x" and "y") to be used for plotting in the argument list of the *plot* function. The graph in Figure 7-2 suggests that the two variables ("petal length" and "petal width") are linearly correlated.

CODE:

```
iris.plot(kind='scatter',x='petal length (cm)',y='petal width (cm)')
```

Output :

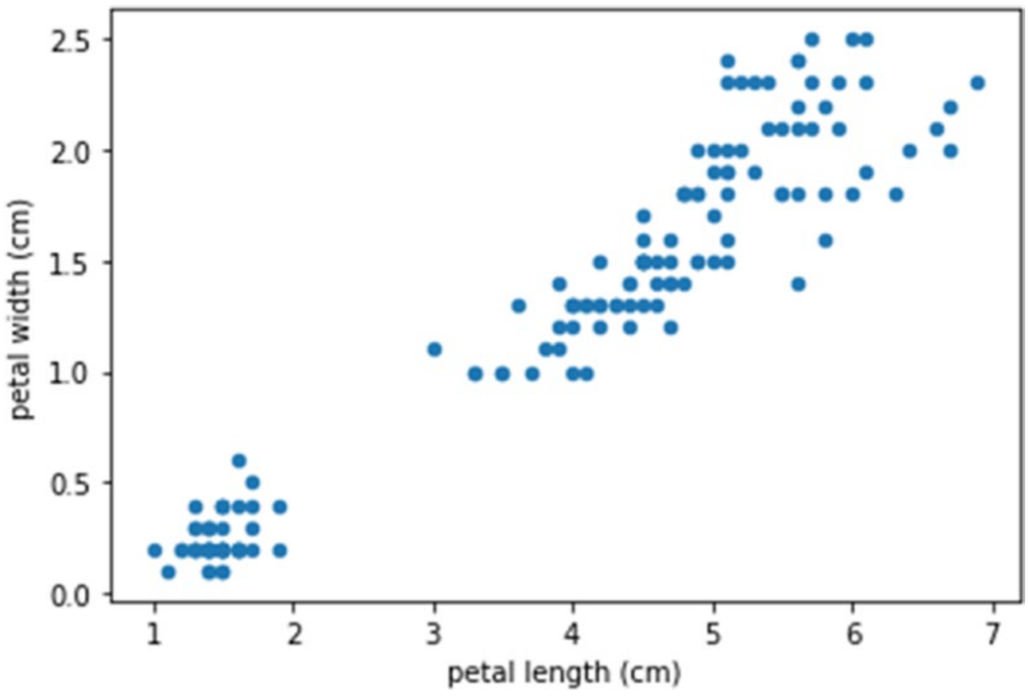


Figure 7-2. *Scatter plot in Pandas*

Histogram

A histogram is used to visualize the frequency distribution with various bars representing the frequencies across various intervals (Figure 7-3). The value 'hist' is used with the *kind* parameter in the *plot* function to create histograms.

CODE:

```
iris['sepal width (cm)'].plot(kind='hist')
```

Output:

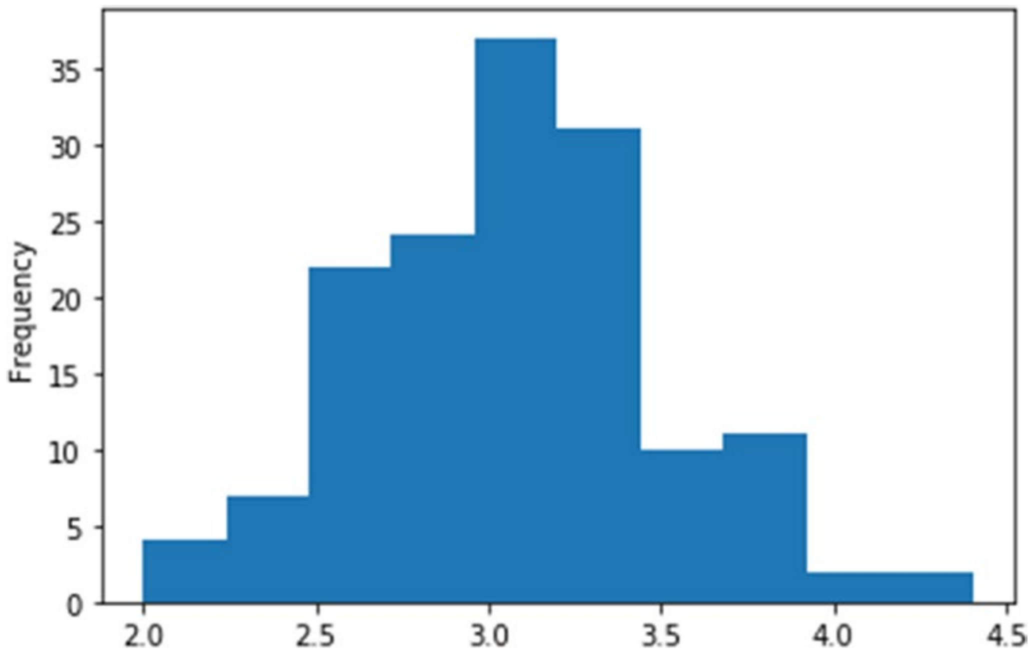


Figure 7-3. *An example of a histogram*

As we can see from this histogram, the “sepal width” variable is normally distributed approximately.

Pie charts

A pie chart shows different values that form a variable as sectors in a circle (Figure 7-4). Note that Pandas requires the *value_counts* function to calculate the number of values in each category as aggregation is not performed on its own when plotting is done in Pandas (we will later see that aggregation is taken care of if plotting is done using the Seaborn library). We need to use the value “pie” with the *kind* parameter to create pie charts.

CODE:

```
iris['species'].value_counts().plot(kind='pie')
```

Output:

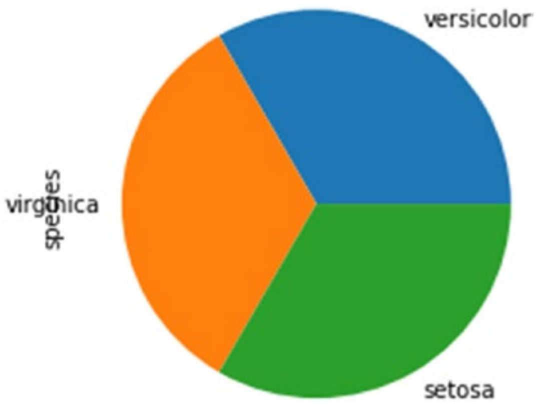


Figure 7-4. *Creating a pie chart with Pandas*

We see that the three species ("virginica", "setosa", and "versicolor") form equal parts of a circle, that is, they have the same number of values.

The Pandas *plot* method is very intuitive and easy to use. By merely changing the value of the *kind* parameter, we can plot a variety of graphs.

Further reading: See more about the kinds of plots that can be used in Pandas:

https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html#other-plots

Seaborn library

Seaborn is another Python-based data visualization library. Seaborn changes the default properties of Matplotlib to adjust the color palettes and perform aggregation automatically on columns. The default settings make it easier to write the code needed for creating various plots.

Seaborn offers the ability to customize these plots as well, but the customization options are less as compared to Matplotlib.

Seaborn enables the visualization of data in more than two dimensions. It also requires data to be in the long (tidy) format, which is in contrast to Pandas, which needs data to be in a wide form.

Let us see how to plot graphs using Seaborn with the *Titanic* dataset.

We use the functions in Seaborn to create different plots for visualizing different variables in this dataset.

The Seaborn library needs to be imported first before its functions can be used. The alias for the Seaborn library is *sns*, which is used for invoking the plotting functions.

CODE:

```
import seaborn as sns
titanic=pd.read_csv('titanic.csv')
```

Box plots

A box plot gives an idea of the distribution and skewness of a variable, based on statistical parameters, and indicates the presence of outliers (denoted by circles or dots), as shown in Figure 7-5. The *boxplot* function in Seaborn can be used to create box plots. The column name of the feature to be visualized is passed as an argument to this function.

CODE:

```
sns.boxplot(titanic['Age'])
```

Output:

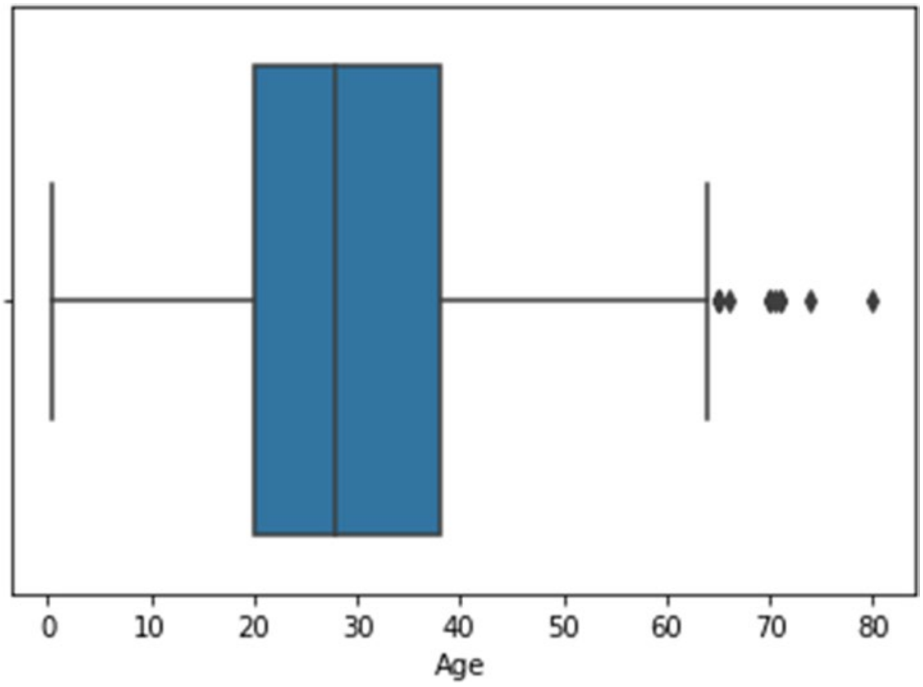


Figure 7-5. *Box plot demonstrating the "Age" variable*

Adding arguments to any Seaborn plotting function

There are two methods we can use when we pass arguments to any function used in Seaborn:

- We can either use the full column name (that includes the name of the DataFrame), skipping the *data* parameter.

  CODE:

  ```
  sns.boxplot(titanic['Age'])
  ```

- Or, mention the column names as strings and use the *data* parameter to specify the name of the DataFrame.

  CODE:

  ```
  sns.boxplot(x='Age',data=titanic)
  ```

Kernel density estimate

The kernel density estimate is a plot for visualizing the probability distribution of a continuous variable, as shown in Figure 7-6. The *kdeplot* function in Seaborn is used for plotting a kernel density estimate.

CODE:

```
sns.kdeplot(titanic['Age'])
```

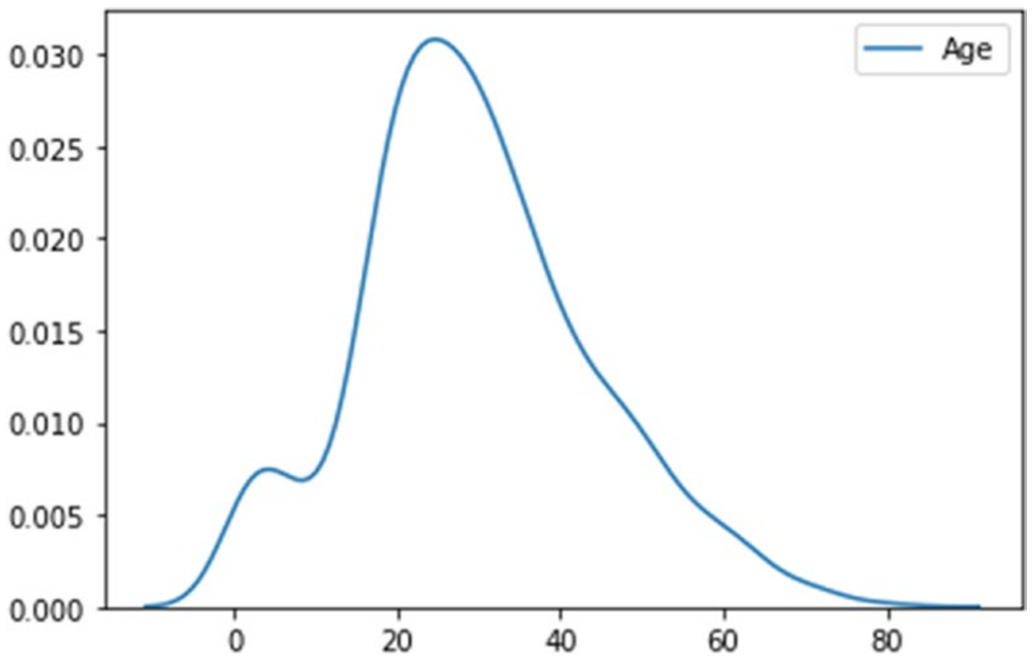


Figure 7-6. *An example of a kernel density estimate (KDE) plot*

Further reading: `https://seaborn.pydata.org/generated/seaborn.kdeplot.html`

Violin plot

A violin plot merges the box plot with the kernel density plot, with the shape of the violin representing the frequency distribution, as shown in Figure 7-7. We use the *violinplot* function in Seaborn for generating violin plots.

CODE:

```
sns.violinplot(x='Pclass',y='Age',data=titanic)
```

Output:

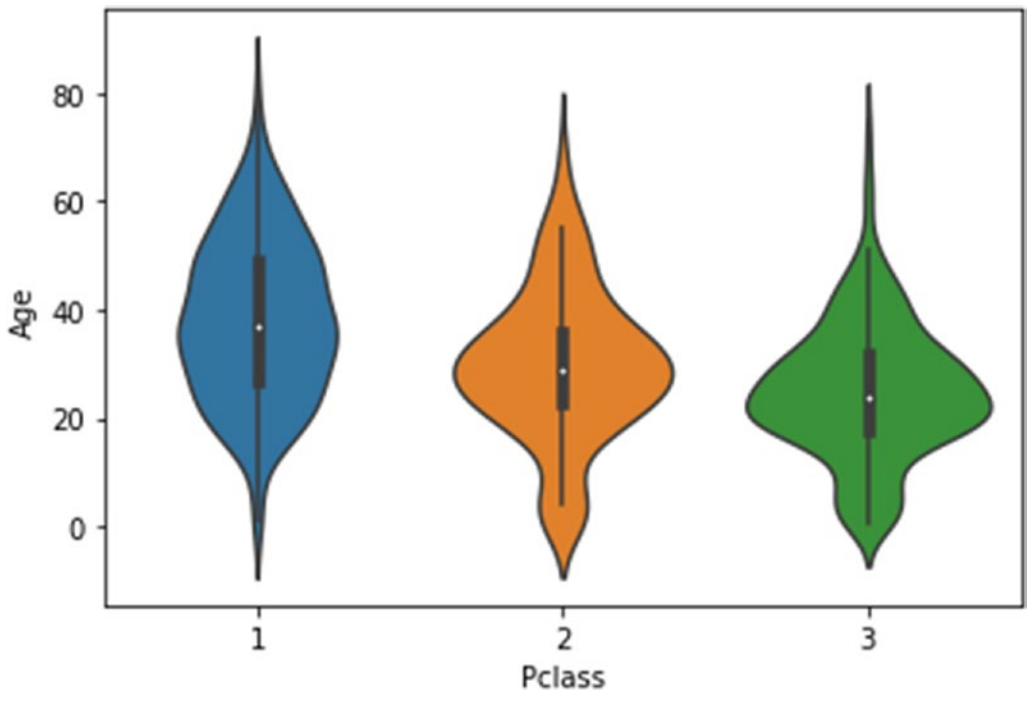


Figure 7-7. *An example of a violin plot in Seaborn*

Further reading: `https://seaborn.pydata.org/generated/seaborn.violinplot.html`

Count plots

Count plots are used to plot categorical variables, with the length of the bars representing the number of observations for each unique value of the variable. In Figure 7-8, the two bars are showing the number of passengers who did not survive (corresponding to a value of 0 for the "Survived" variable) and the number of passengers who survived (corresponding to a value of 1 for the "Survived" variable). The *countplot* function in Seaborn can be used for generating count plots.

CODE:

```
sns.countplot(titanic['Survived'])
```

Output:

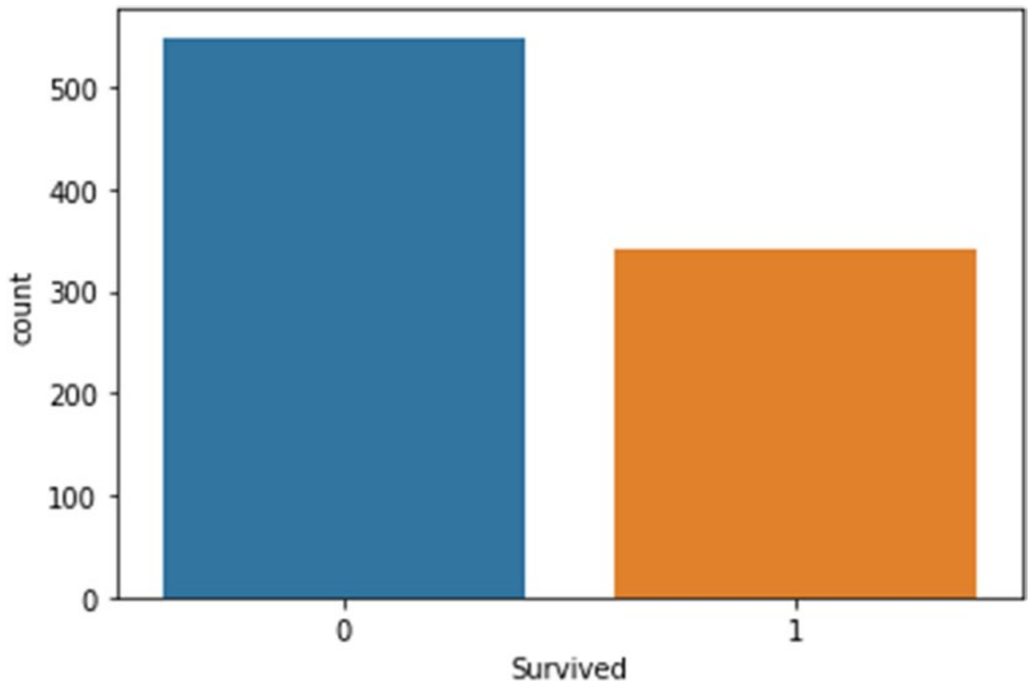


Figure 7-8. *An example of a countplot in Seaborn*

Further reading: `https://seaborn.pydata.org/generated/seaborn.countplot.html`

Heatmap

A heatmap is a graphical representation of a correlation matrix, representing the correlation between different variables in a dataset, as shown in Figure 7-9. The intensity of the color represents the strength of the correlation, and the values represent the degree of correlation (a value closer to one represents two strongly correlated variables). Note that the values along the diagonal are all one since they represent the correlation of the variable with itself.

The *heatmap* function in Seaborn creates the heat map. The parameter *annot* (with the value "True") enables the display of values representing the degree of correlation, and the *cmap* parameter can be used to change the default color palette. The *corr* method creates a DataFrame containing the degree of correlation between various pairs of variables. The labels of the heatmap are populated from the index and column values in the correlation DataFrame (titanic.corr in this example).

CODE:

```
sns.heatmap(titanic.corr(),annot=True,cmap='YlGnBu')
```

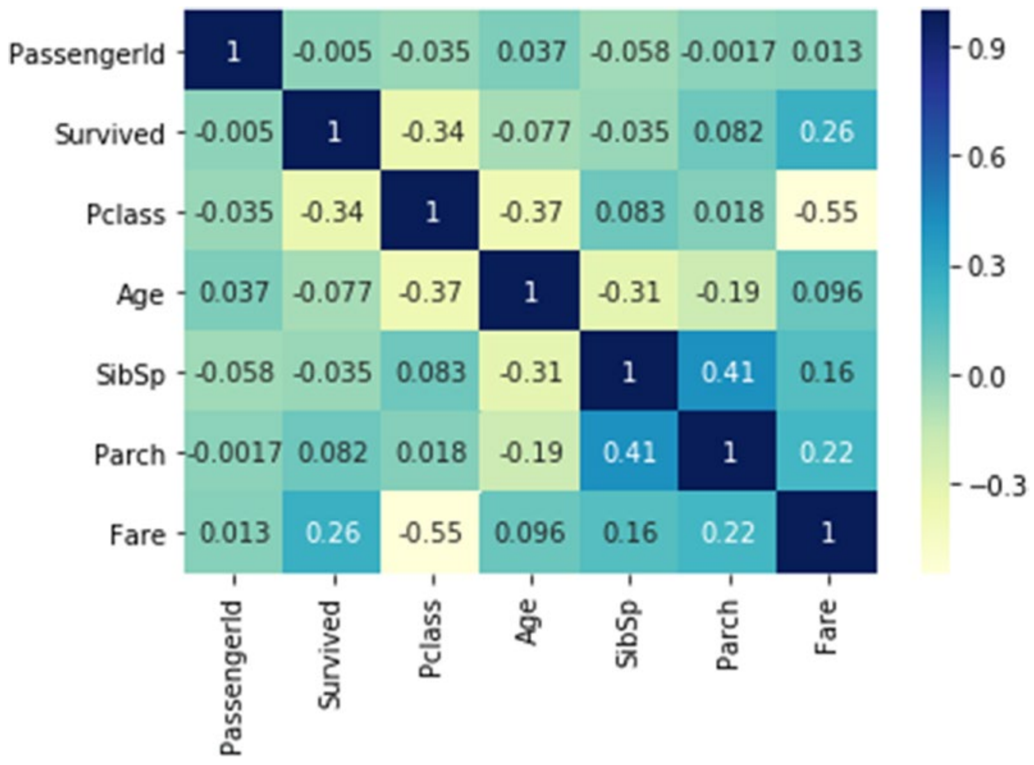


Figure 7-9. *An example of a heatmap in Seaborn*

Further reading:

See more about the "heatmap" function and its parameters:

https://seaborn.pydata.org/generated/seaborn.heatmap.html

See more about customizing color palettes and color maps: https://seaborn.pydata.org/tutorial/color_palettes.html

Facet grid

A facet grid represents the distribution of a single parameter or the relationship between parameters across a grid containing a *row*, *column*, or *hue* parameter, as shown in Figure 7-10. In the first step, we create a grid object (the *row*, *col*, and *hue* parameters are optional), and in the second step, we use this grid object to plot a graph of our choice (the name of the plot and the variables to be plotted are supplied as arguments to the map function). The *FacetGrid* function in Seaborn is used for plotting a facet grid.

Example:

CODE:

```
g = sns.FacetGrid(titanic, col="Sex",row='Survived') #Initializing the grid
g.map(plt.hist,'Age')#Plotting a histogram using the grid object
```

Output:

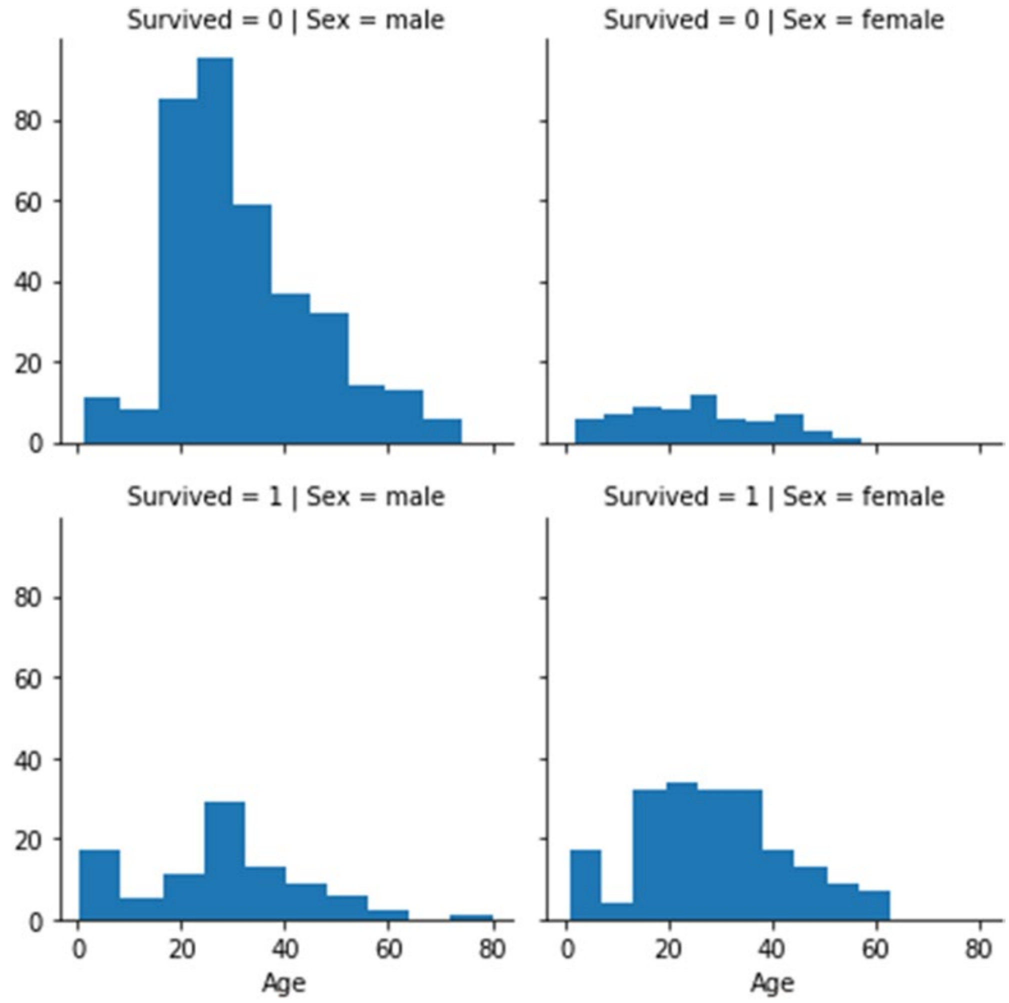


Figure 7-10. *An example of a facet grid*

Regplot

This plot uses the linear regression model to plot a regression line between the data points of two continuous variables, as shown in Figure 7-11. The Seaborn function *regplot* is used for creating this plot.

CODE:

```
sns.regplot(x='Age',y='Fare',data=titanic)
```

Output:

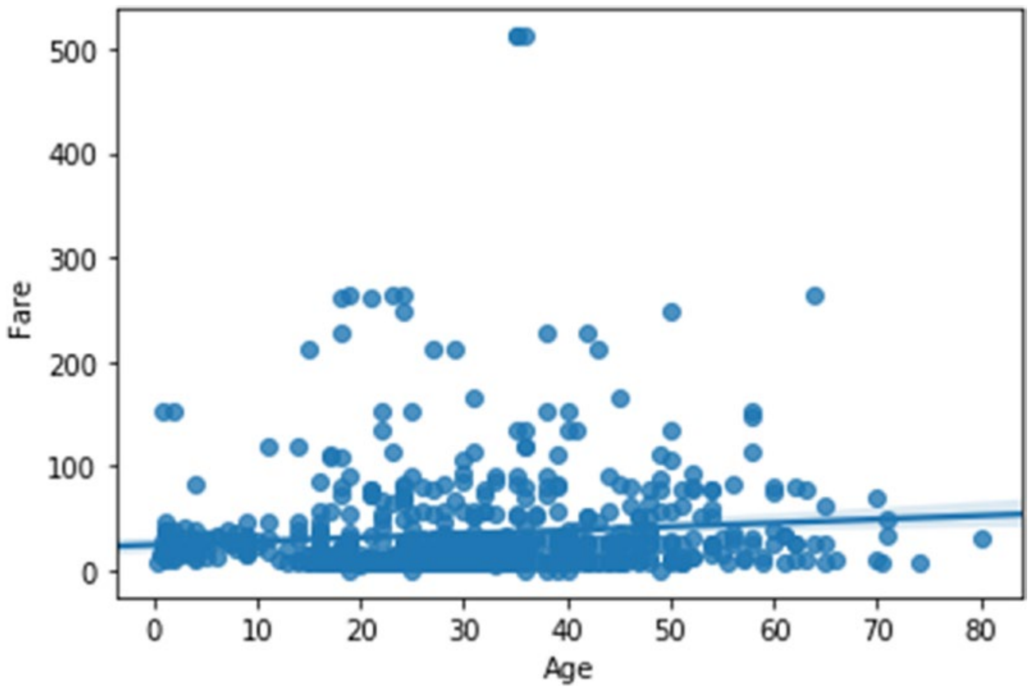


Figure 7-11. *An example of a regplot*

Further reading:

https://seaborn.pydata.org/generated/seaborn.regplot.html#seaborn.regplot

lmplot

This plot is a combination of a regplot and a facet grid, as shown in Figure 7-12. Using the *lmplot* function, we can see the relationship between two continuous variables across different parameter values.

In the following example, we plot two numeric variables ("Age" and "Fare") across a grid with different row and column variables.

CODE:

```
sns.lmplot(x='Age',y='Fare',row='Survived',data=titanic,col='Sex')
```

Output:

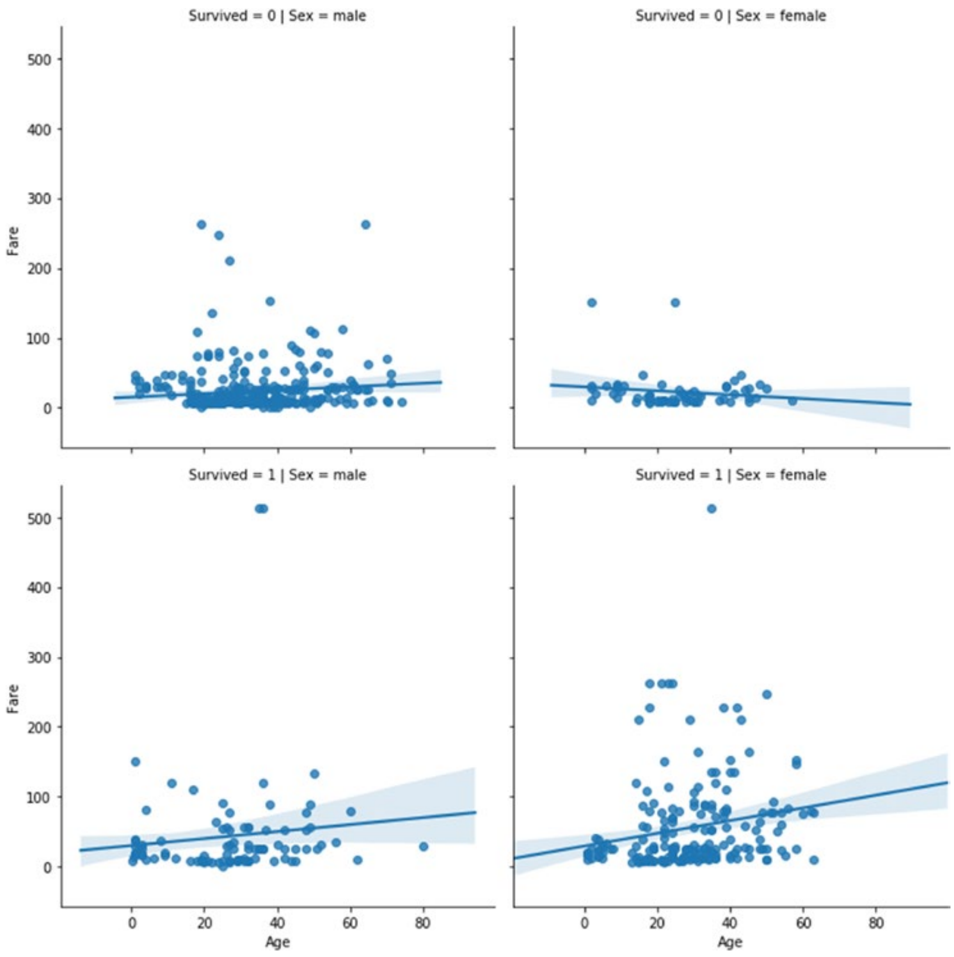


Figure 7-12. *An example of lmplot*

The following summarizes the differences between regplot and lmplot:

- The *regplot* function takes only two variables as arguments, whereas the lmplot function accepts multiple arguments.
- The *lmplot* function works at the level of the figure object, while the regplot function works at the level of an axes object.

Further reading on the lmplot: `https://seaborn.pydata.org/generated/seaborn.lmplot.html`

See more on the differences between regplot and lmplot: `https://seaborn.pydata.org/tutorial/regression.html#functions-to-draw-linear-regression-models`

Strip plot

A strip plot is similar to a scatter plot. The difference lies in the type of variables used in a strip plot. While a scatter plot has both variables as continuous, a strip plot plots one categorical variable against one continuous variable, as shown in Figure 7-13. The Seaborn function *striplot* generates a strip plot.

Consider the following example, where the "Age" variable is continuous, while the "Survived" variable is categorical.

CODE:

```
sns.stripplot(x='Survived',y='Age',data=titanic)
```

Output:

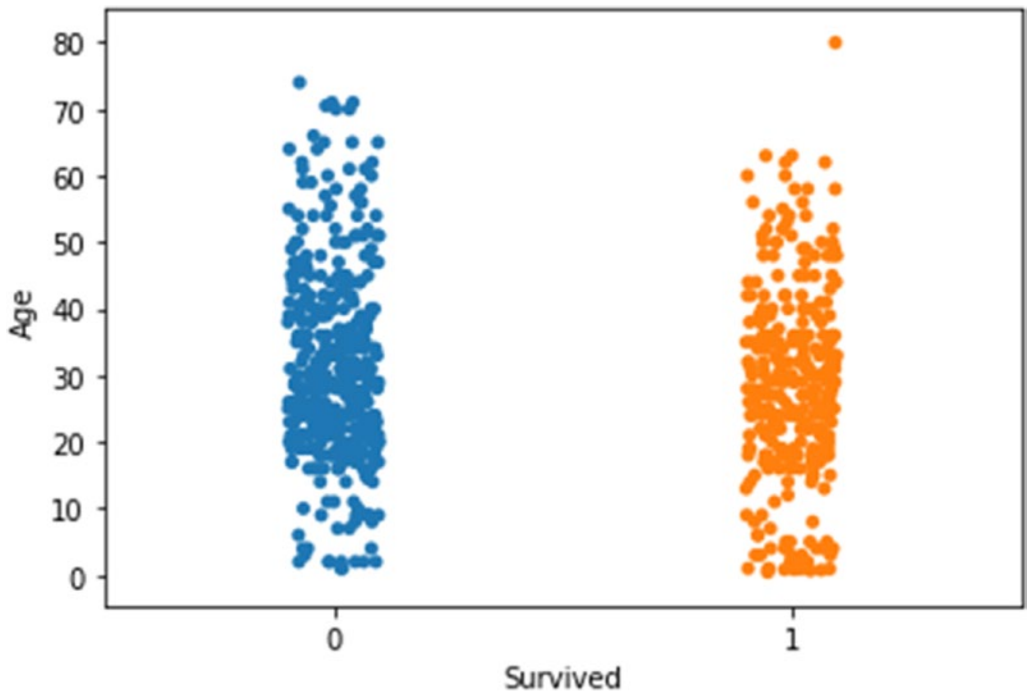


Figure 7-13. *An example of a strip plot*

Further reading: https://seaborn.pydata.org/generated/seaborn.stripplot.html

Swarm plot

A swarm plot is similar to a strip plot, the difference being that the points in a swarm plot are not overlapping like those in a strip plot. With the points more spread out, we get a better idea of the distribution of the continuous variable, as shown in Figure 7-14. The Seaborn function *swarmplot* generates a swarm plot.

CODE:

```
sns.swarmplot(x='Survived',y='Age',data=titanic)
```

Output:

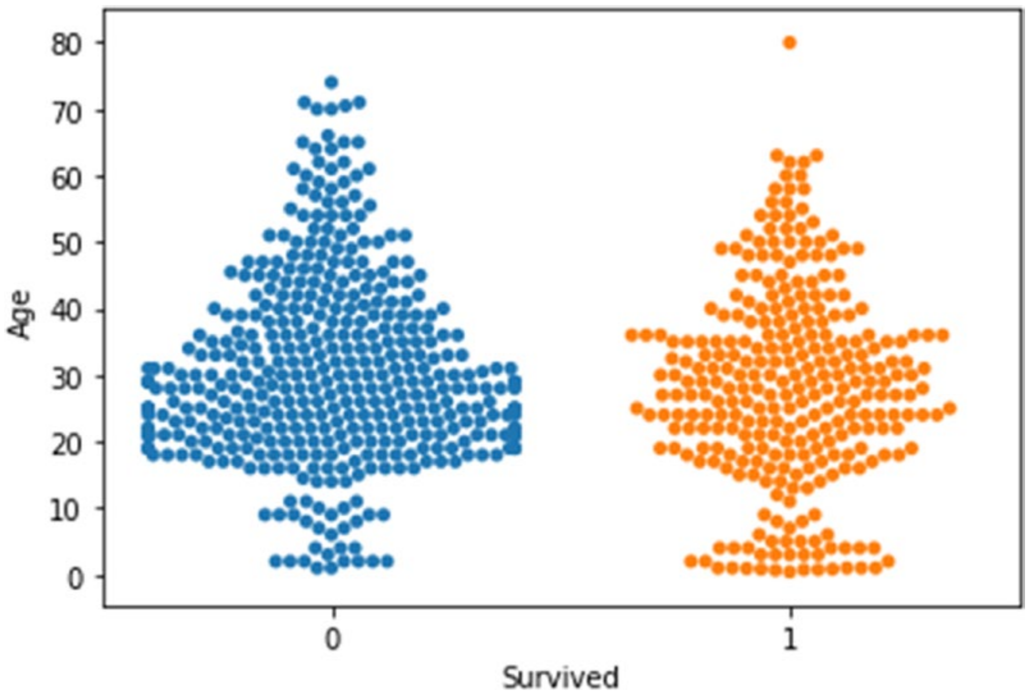


Figure 7-14. *An example of a swarm plot*

Further reading: https://seaborn.pydata.org/generated/seaborn.swarmplot.html#seaborn.swarmplot

Catplot

A catplot is a combination of a strip plot and a facet grid. We can plot one continuous variable against various categorical variables by specifying the *row*, *col*, or *hue* parameters, as shown in Figure 7-15. Note that while the strip plot is the default plot generated by the *catplot* function, it can generate other plots too. The type of plot can be changed using the *kind* parameter.

CODE:

```
sns.catplot(x='Survived',y='Age',col='Survived',row='Sex',data=titanic)
```

Output:

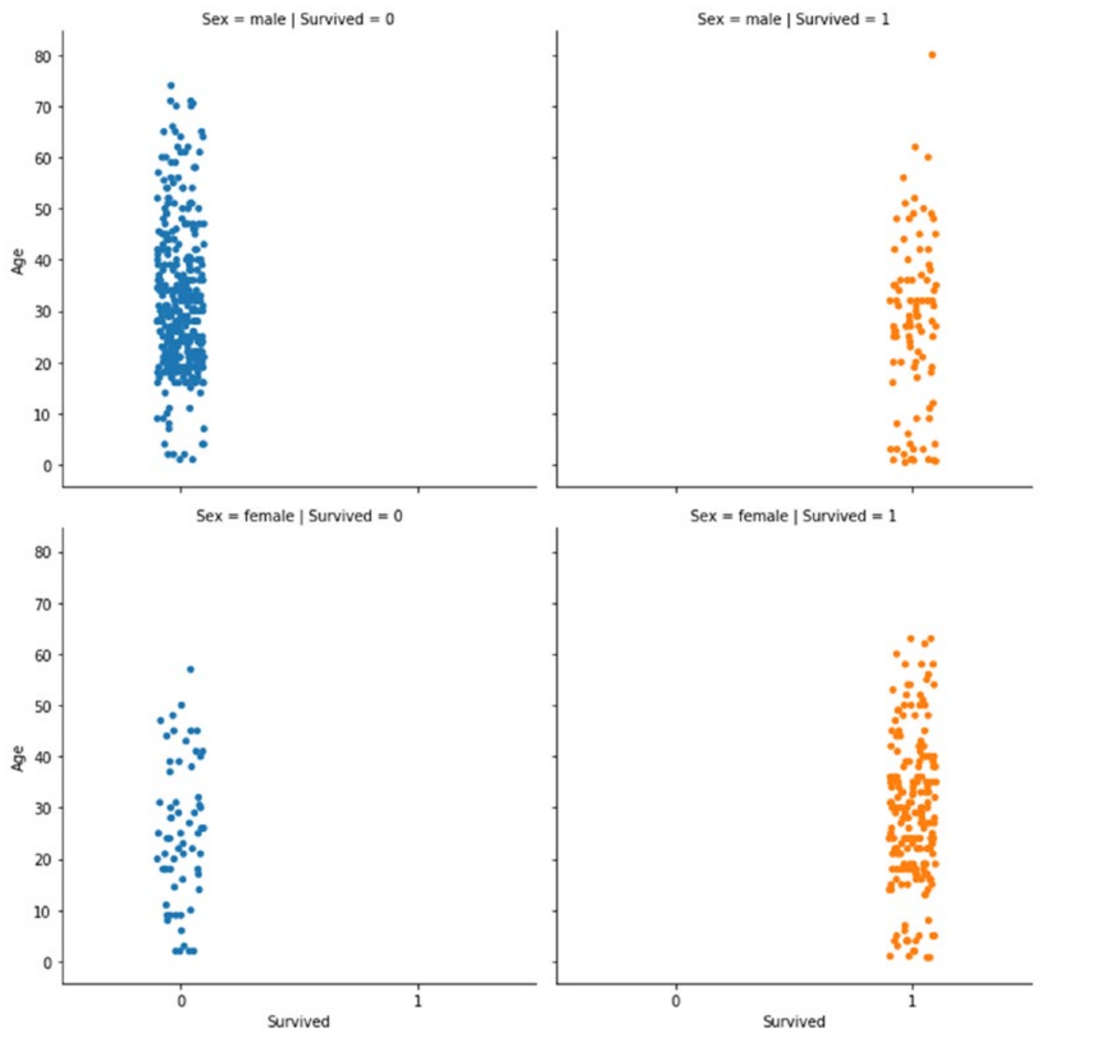


Figure 7-15. *An example of a catplot in Seaborn*

Further reading: https://seaborn.pydata.org/generated/seaborn.catplot.html

Pair plot

A pair plot is one that shows bivariate relationships between all possible pairs of variables in the dataset, as shown in Figure 7-16. The Seaborn function *pairplot* creates a pair plot. Notice that you do not have to supply any column names as arguments since all the variables in the dataset are considered automatically for plotting. The only parameter that you need to pass is the name of the DataFrame. In some of the plots displayed as part of the pair plot output, any given variable is also plotted against itself. The plots along the diagonal of a pair plot show these plots.

CODE:

```
sns.pairplot(data=titanic)
```

Output:

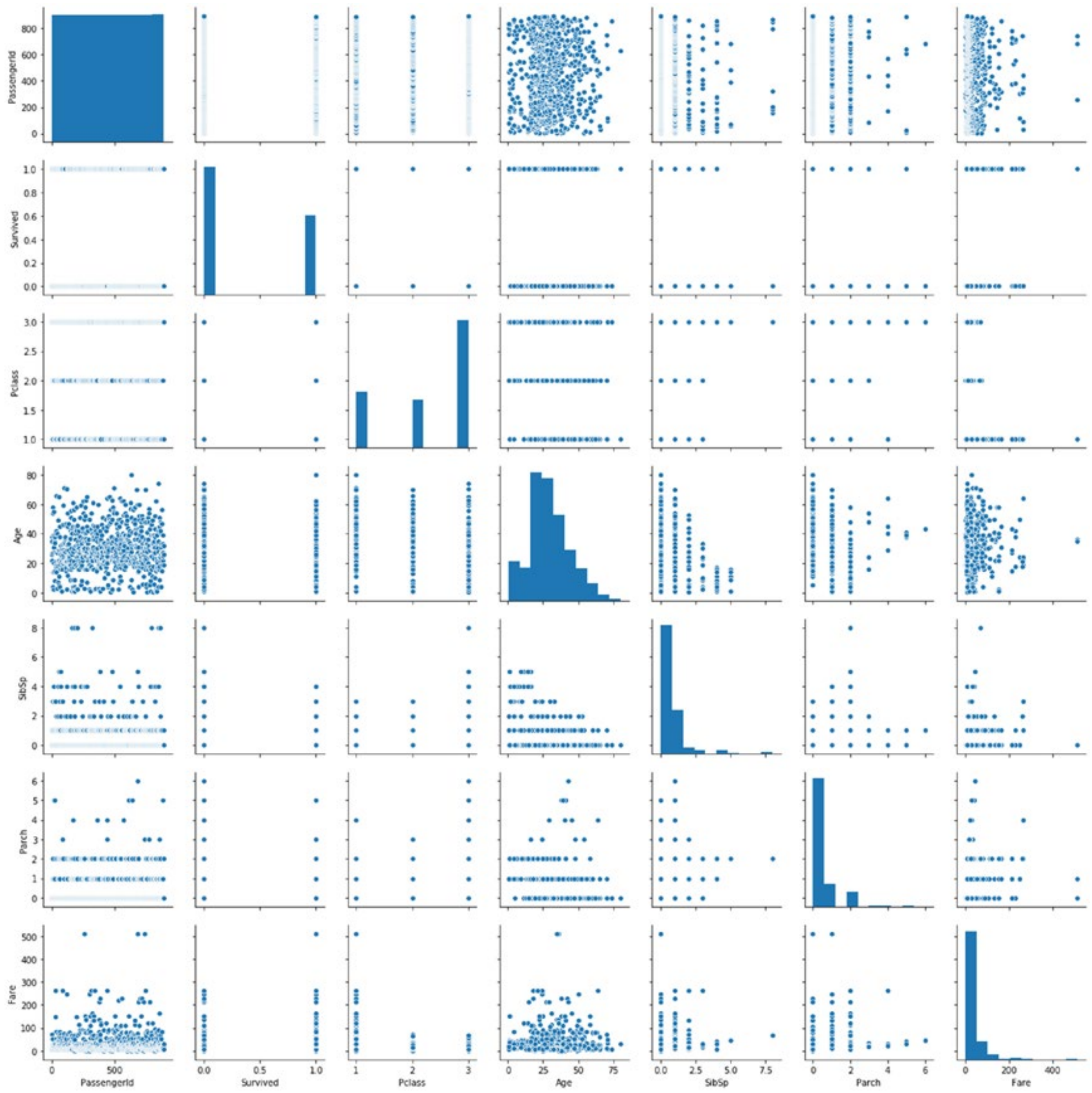


Figure 7-16. *An example of a pair plot in Seaborn*

Joint plot

The joint plot displays the relationship between two variables as well as the individual distribution of the variables, as shown in Figure 7-17. The *jointplot* function takes the names of the two variables to be plotted as arguments.

CODE:

```
sns.jointplot(x='Fare',y='Age',data=titanic)
```

Output:

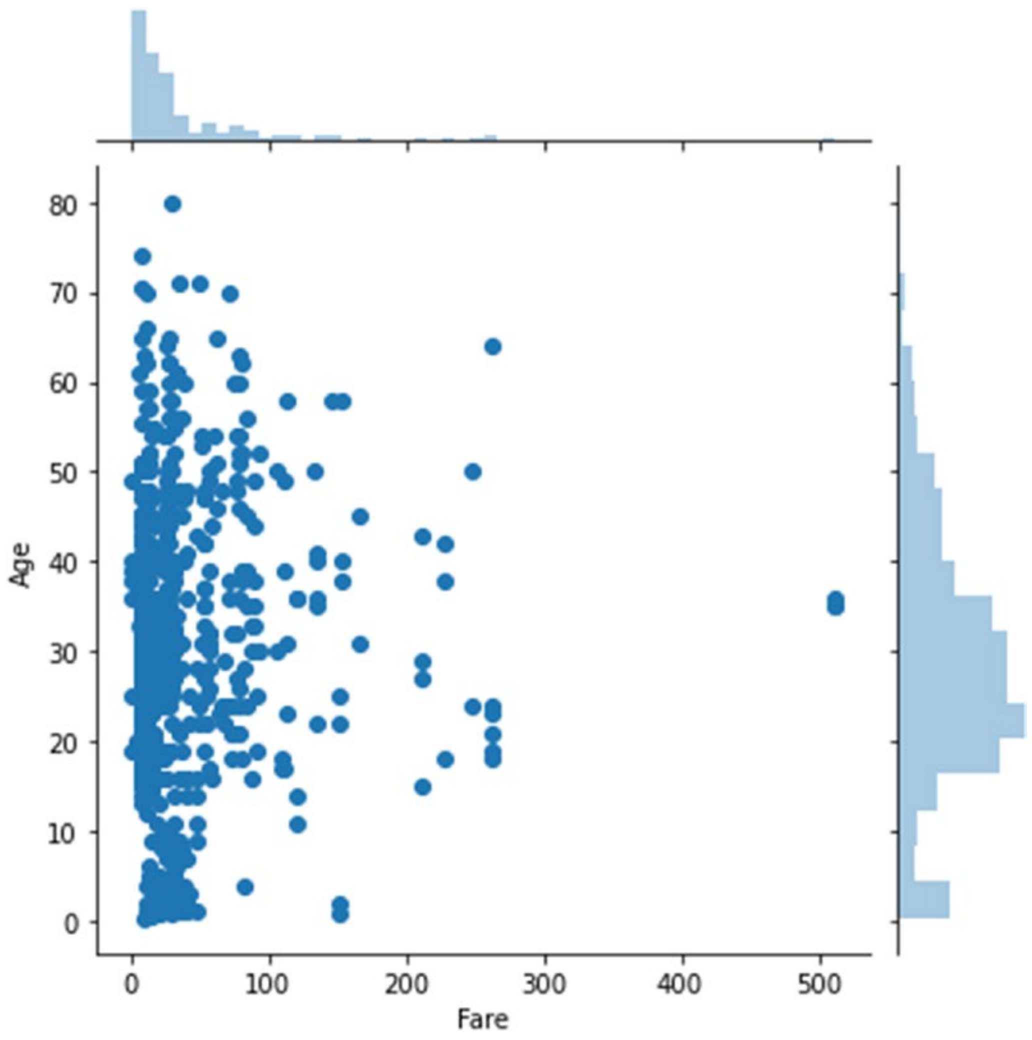


Figure 7-17. *An example of a jointplot in Seaborn*

Further reading on the jointplot: https://seaborn.pydata.org/generated/seaborn.jointplot.html#seaborn.jointplot

Further reading on the Seaborn library:

Examples of various graphs that can be created in Seaborn: https://seaborn.pydata.org/examples/index.html

Getting started and solved examples in Seaborn: https://seaborn.pydata.org/introduction.html

Summary

1. Three Python-based libraries can be used for visualization in Python – Matplotlib (which is based on Matlab), Pandas, and Seaborn.
2. Before we draw graphs, we need to figure out the type of variable that needs to be plotted and the number of variables that need to be plotted. Use bar charts and pie charts for categorical variables, and histograms and scatter plots for continuous variables.
3. Matplotlib has two interfaces that are used for plotting – the stateful interface and the object-oriented interface. The stateful interface uses the *pyplot* class and keeps track of the current state of the object of this class. The object-oriented interface uses a hierarchy of objects to represent various elements of the plot and uses these objects for plotting.
4. The *plot* function, which is used in Pandas for plotting, uses Matplotlib at the back end. This function makes it easy to draw any kind of graph just by changing the arguments passed to it, thus utilizing the principle of polymorphism (one name, many forms).
5. Seaborn is another library that uses Matplotlib at the back end. By changing its default parameters, it minimizes the need to perform aggregations, label, and color code the elements of your graph. It also provides the ability to visualize more than two variables.

In the next chapter, we examine some real-world case studies where we will put into practice what we have learned about visualization and data wrangling.

Review Exercises

Question 1

Which plot would you use when you have the following variables?

- One categorical variable
- One continuous variable
- Two continuous variables
- Three or more continuous variables
- One continuous and one categorical variable
- Two continuous and two or more categorical variables

Question 2

Match the functions on the left with the correct description on the right

1. Facet grid	a. Plot showing relationships between all possible pairs of variables and distributions of individual variables
2. Catplot	b. Plot of a continuous variable across a grid of categorical parameters
3. Swarm plot	c. Plot of one continuous and one categorical variable, with points not overlapping
4. Pair plot	d. Plot that combines a box plot with a kernel density estimate
5. Violin plot	e. Combination of a facet grid and strip plot

Question 3

Which among the following is true about the visualization performed using the Pandas library?

1. Aggregation is performed automatically while plotting graphs
2. Pandas requires data in the long format
3. The *plot* method is used to plot graphs
4. The *type* parameter is used to specify the type of plot
5. The *kind* parameter to specify the type of plot

Question 4

The magic command needed for displaying Matplotlib and Pandas graphs inline is

1. %matplotlib
2. %inline
3. %matplotlib inline
4. None of the above

Question 5

The axes object refers to the

1. x axis
2. y axis
3. Both the x and y axis
4. The subplot containing the graph

Question 6

For a given DataFrame, df, how do we specify the following parameters used in the heatmap function?

1. Correlation matrix
2. Color map for coloring the squares in a heatmap
3. The numeric value of the degree of correlation between each of the parameters

Question 7

The Sklearn library has a built-in dataset, *Iris*. It contains samples from various species of the iris plant. Each sample contains four attributes: sepal length, sepal width, petal length, petal width, and species (*Iris setosa*, *Iris versicolor*, and *Iris virginica*), and there are 50 samples of each species.

- Read the data from this dataset into a DataFrame.
- Create a 10*5 figure with two subplots.
- In the first subplot, plot the petal length vs. petal width.

- In the second subplot, plot the sepal length vs. sepal width.
- For each of the plots, label the x and y axes and set a title.

Question 8

Load the data from tips dataset (built into Seaborn) using the load_dataset function in Seaborn. This dataset contains the total bill amounts and the tip values for different customers visiting a restaurant. The customers are categorized according to their gender, smoking preference, and the day and time of their visit.

- Create a plot that shows the distribution (strip plot) of the total bill for smokers and nonsmokers, across a grid containing different values for the time and sex columns.
- Create a plot to show the relationship between the "tip" and "total_bill" columns for: males and smokers, males and nonsmokers, females and smokers, and females and nonsmokers.

Answers

Question 1

- One categorical variable: count plot
- One continuous variable: histogram, kernel density estimate
- Two continuous variables: scatter plot, line plot
- Three or more continuous variables: heat map
- One continuous and one categorical variable: strip plot, swarm plot, bar plot
- Two continuous and two or more categorical variables: catplot, facet grid, lmplot

Question 2

1-b; 2-e; 3-c; 4-a; 5-d

Question 3

Options 3 and 5

Pandas uses the *plot* method with the kind parameter to create various graphs. Pandas requires data to be in the wide or aggregated form. Aggregation is not done by default, unlike Seaborn. The *value_counts* method is required for aggregation before the *plot* method is applied.

Question 4

Option 3

We use the magic command (*%matplotlib inline*) for displaying graphs inline in Matplotlib and Pandas.

Question 5

Option 4

The term "axes" is a misnomer and does not refer to the x or y axis. It refers to the subplot or plotting area, which is a part of the figure object. A figure can contain multiple subplots.

Question 6

- Correlation matrix: df.corr()(generated a DataFrame representing the correlation matrix)
- Color map for coloring the squares in a heatmap: *cmap* parameter
- Denoting the numeric value of the degree of correlation: annot parameter (*annot=True*)

Question 7

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd
#importing the iris dataset
data=load_iris()
iris=pd.DataFrame(data=data.data,columns=data.feature_names)
iris['species']=data.target
iris['species']=iris['species'].map({0:'setosa',1:'versicolor',2:'virginica'})
iris['species'].value_counts().plot(kind='bar')
fig=plt.figure(figsize=(10,5))
```

```
ax1=fig.add_subplot(121) #defining the first subplot
#plotting petal length vs petal width in the first subplot
iris.plot(kind='scatter',x='petal length (cm)',y='petal width (cm)',ax=ax1)
#adding the title for the first subplot
ax1.set_title("Petal length vs width")
#adding the label for the X axis
ax1.set_xlabel("Petal length")
#adding the label for the Y axis
ax1.set_ylabel("Petal width")
ax2=fig.add_subplot(122) #defining the second subplot
#plotting sepal length vs sepal width in the second subplot
iris.plot(kind='scatter',x='sepal length (cm)',y='sepal width (cm)',ax=ax2)
ax2.set_xlabel("Sepal length")
ax2.set_ylabel("Sepal width")
ax2.set_title("Sepal length vs width")
#Increasing the distance width between the subplots
fig.subplots_adjust(wspace=1)
```

Question 8

```
import seaborn as sns
#loading the data into a DataFrame using the load_dataset function
tips = sns.load_dataset("tips")
#creating a catplot for showing the distribution of the total bill for
different combinations of parameters
sns.catplot(x='smoker',y='total_bill',row='time',col='sex',data=tips)
#defining the FacetGrid object and setting the row and column values
g=sns.FacetGrid(tips,row='sex',col='smoker')
#specifying the plot and the columns we want to display
g.map(plt.scatter,'tip','total_bill')
```